Towards Bridging Online Game Playing and Live Broadcasting: Design and Optimization

Ryan Shea, Di Fu, and Jiangchuan Liu
Simon Fraser University
Burnaby, Canada
{rws1, dif, jcliu}@cs.sfu.ca

ABSTRACT

Recent years have witnessed the emergence and growth of Cloud Gaming, where players interact with the remote game instance and receive rendered game scenes in video stream. Meanwhile, broadcasting and viewing games through live streaming platforms, e.g., Twitch.tv, have become increasingly popular. The interaction and performance of the many modules involved in this new generation of gaming and streaming platforms have yet to be closely investigated. In this paper, we present an initial experiment-based performance study, in which we profile the architecture of realworld gaming and streaming platforms, namely the Open Broadcast Software (OBS) module and its connection to the Twitch server. Our investigation shows that the recording operation can greatly increase the CPU utilization and the power consumption can increase over 60% on the game streaming computer. The use of advanced hardware encoding found on modern GPUs can greatly alleviate these performance issues. Yet, through profiling, we show that hardware encoding can introduce remarkable delays to the whole pipeline. We track this to a complicated interplay between the CPUs power saving methods and the implementation of hardware encoders.

Categories and Subject Descriptors

H.5.1 [**INFORMATION INTERFACES AND PRESENTATION**]: Multimedia Information Systems

General Terms

Design, Performance, Measurement

Keywords

Cloud Gaming, Graphics Processing Unit

1. INTRODUCTION

Live broadcasting video games has seen an amazing increase in popularity in the last few years. A large factor of this growth is fuelled by the rising popularity of organized video game competitions often referred to as `e-sports`. It is now estimated that over 30 million Americans watch or participate in competitive e-sports [1]. This rapid expansion has also attracted the interest of major corporate sponsors such as Coca-Cola and HBO.

One of the largest and most successful live broadcasting systems is provided by `Twitch.tv`. Twitch now attracts over 1 million monthly active broadcasters and 45 million monthly viewers [7]. Gamer player can live stream to Twitch users through such broadcasting software as the widely deployed open-source OBS (Open Broadcast Software).[1] There have been significant studies on online gaming and on live streaming. The combination of them opens many more opportunities but also new challenges. The interaction and performance of the many modules involved in this new generation of gaming and streaming platforms have yet to be closely investigated.

In this paper, we present an initial experiment-based performance study and profile the architecture of realworld gaming and streaming platforms, particularly on the OBS module and its connection to the Twitch server. Our investigation of live game streaming shows that the recording operation can greatly increase the CPU utilization and the power consumption can increase over 60% on the game streaming computer. In the worst case, OBS can drop the frame rate of some game scenes by over 20%. The use of advanced hardware encoding found on modern GPUs can greatly alleviate these performance issues. Yet, through profiling, we show that due to *blocking calls*, hardware encoding can face significant delays in the whole pipeline. We track this to a complicated interplay between the CPUs power saving methods and the implementation of hardware encoders. We provide a simple modification to OBS, which can improve the performance without greatly increasing the power consumption of the system. Finally, using what we learned from profiling, we show that broadcastings can be enhanced by integration with a public cloud and a real world cloud gaming platform. Our initial experiments on this hybrid platform indicate that cloud-gaming and live game streaming can be combined with only a 2% cost to performance of the system.

2. GAME STREAMING ARCHITECTURE

In this section, we explore the framework of a real world game streaming service using `Twitch`, the most popular gameplay streaming site as a case study. We begin our

[1] `https://obsproject.com/`

NOSSDAV '15 March 18-20, 2015, Portland, OR, USA
Copyright 2015 ACM 978-1-4503-3352-8/15/03$15.00
http://dx.doi.org/10.1145/2736084.2736089

discussion by looking inside the Open Broadcast Software (OBS), a key component for Twitch-like game streaming. OBS is widely deployed and and one of the top recommended software for the Twitch streaming service. Since the OBS project is open-source, it allows us to uncover the framework of the system, as well as measure performance issues using profiling techniques and finally implement improvements.

2.1 OBS Architecture

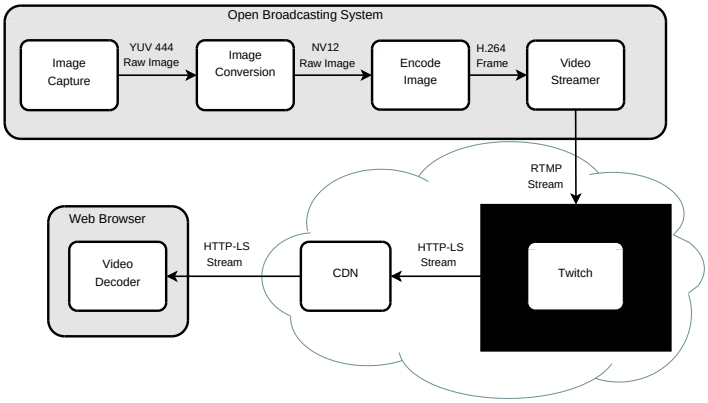

Figure 1: Game Capture and Streaming Pipeline

Through systematic source code and runtime analysis, we have produced a high-level framework of the OBS system, shown in Figure 1. It tracks the game image frame from its capture, to its transmission to Twitch, and then finally to the end-user.

As can be seen, OBS first captures the image in YUV444 format. Using advanced graphic hooking techniques, the user can either capture the whole desktop or a specific window depending on which API the game is programmed in, for example, DirectX or OpenGL. The image will be down sampled a raw YUV420 (NV12) format, which is then fed to an H.264 encoder. On most systems, this is handled by the open source `x264`[2] H.264 software encoding engine, which is highly optimized and supports parallel encoding on multiple CPUs. On newer platforms, OBS also supports H.264 encoding with not only the `x264` software encoder but also NVIDIA's advanced `NVENC`[3] hardware encoder, thereby offloading the computationally expensive encoding operation to a dedicated piece of hardware. We will discuss hardware versus software encoders in the next section.

After the image is encoded, OBS sends the frame to the streaming engine, which for Twitch is an implementation of the RTMP (Real-Time Messaging Protocol) streaming protocol. We have captured packets at both the game streamer side and the viewer side and performed a protocol analysis. In our case, the RTMP packets are sent from our local client to a Twitch server located in California, which converts the stream into a HTTP Live Stream (HLS). This conversion can include transcoding as well as distribution to content distribution networks (CDNs) in different regions. For our purposes, we treat Twitch as a black-box, though more information on certain internal architecture of Twitch can be obtained from an interview with a Twitch founder.[4]

[2] `http://www.videolan.org/developers/x264.html`
[3] `http://developer.download.nvidia.com/compute/nvenc/`
[4] `http://highscalability.com/blog/2010/3/16/justintvs-live-video-broadcasting-architecture.html`
[5] `https://unigine.com/products/heaven/`

Encoding	CPU OBS	CPU Total	Energy (Wall)
Game Only	N/A	~40%	157 W
SW (30 FPS)	~37%	~79%	250 W
SW (60 FPS)	~57%	~100%	244 W
HW (30 FPS)	~10%	~50%	158 W
HW (60 FPS)	~15%	~55%	183 W

Table 1: CPU and Energy Consumption

Figure 2: Amperage Measurement

3. GAME STREAMING PERFORMANCE

3.1 Measurement Platform Configuration

Our streaming server contains an Intel Haswell Xeon E3-1245 quad core processor and the motherboards chip-set is Intel C226. The server has 8 GB of 1600 MHz DDR-3 memory installed. Networking is provided through a Intel i217LM 1000 Mb/s ethernet card. We have also installed a NVIDIA GTX 970 with 4 GB of GDDR5 memory, a representative for advanced GPUs on the market that support hardware H.264 encoding. The operating system is Windows Server 2008 64 bit edition, with the latest service packs and DirectX versions installed. We start our discussion with the power consumption of different encoding methods.

3.2 CPU Usage and Power Consumption

As we intend to measure energy consumption of the system using hardware or software encoding, we employed a digital multimeter with PC-link capability, namely the MASTECH MS2115B clamp meter (precision of ± 2.5%). It was used with an AC-line splitter so we could directly record the input current of the power supply, as can be seen in Figure 2. We recorded samples every 500 ms from our multi-meter using the PC-link over USB, and also recorded the CPU usage of the OBS executable every second for the duration of the test. For a capture target, we utilized the Unigine benchmark Heaven[5], which supports graphics rendering techniques such as DirectX 11 and OpenGL 4.0. We used a 1920 x 1080 resolution, with all high detail settings enabled, and configured OBS to record the heaven benchmark and stream it to Twitch. We set both the software x264 and the NVENC based hardware encoder to the optimal settings for streaming to Twitch at 1080p. That is, using constant bit rate encoding (CBR), at a bit-rate of 3500 kb/s, and a key-frame interval of 2 seconds. For

both the software and hardware encoders, we recorded and streamed an entire run of the benchmark, recording at both 30 frames per second (FPS) and 60 FPS, and collecting the average energy consumption and processor usage over the run.

The results for this experiment are given in Table 1. We first report the CPU consumption and energy consumption of the heaven benchmark running on the system with no recording or streaming. As can be seen, processing and rendering the heaven benchmark on our system require approximately 40% of the system's available resources, as well as 158 watts at the wall. The power consumption for this game is dominated by the CPU and GPU, since there is no pre-load, and it pre-caches everything from the disk before running. Next, we configured OBS to record at 30 FPS using the x264 encoder, and see a dramatic increase in terms of both the CPU and energy consumption. OBS utilizing the x264 software encoder consumes nearly 37% of the CPU on its own, and the combined usage has doubled to 79%. The power consumption of the system has also greatly risen by a staggering 100 watts, an increase of nearly 60%. We also reconfigured OBS to once again utilize the x264 software encoder but increase the encoding rate to 60 fps. At this setting, the CPU became completely saturated by the encoding task, which takes up nearly 60% of the available CPU and the Heaven benchmark itself. Surprisingly, the total energy consumption in the system drops slightly, from 250 watts in the 30 FPS experiment to 244 watts in 60 FPS experiment. We probed deeper into this interesting anomaly and found it was actually due a reduction in the GPU utilization. The reduction in GPU utilization was mainly due to major task interference between the CPU 3D physics operations of the Heaven benchmark and OBS's x264 encoding engine. This led the GPU to be starved of scenes to render, creating a drop in the number of frames rendered per second by the GPU. We will explore this task interference issue in detail in section 3.3.

We now measure the performance of encoding using the advanced NVENC hardware H.264 encoder. Once again the results can be seen in Table 1. As can be seen, the hardware encoder consumes only 10% of the available CPU. This is a huge improvement over the software x264 encoder, which takes nearly 4 times the amount of CPU to accomplish the same thing. The energy consumption provides yet another interesting result, being nearly identical to the base line energy consumption case. Further, exploration on this issue has led us to find that this happens for two interesting reasons. First, the GPU download phase causes a slight drop in GPU utilization and frame rate, which causes the GPU to slightly reduce its energy consumption. The second reason has to do with the advanced power management features found in modern operating systems and CPUs. Encoding only takes 10% of the CPU for a total of 50% of the total available CPU when we factor in the game logic, as encoding and game logic can be processed on only two of the cores of our 4 core CPU. When compared to the base case of the game running alone, we still consume 40% of the CPU, which still requires 2 cores of our 4 core platform. We have confirmed this by looking at the Windows power-management system and watching the cores enter their sleep states (c-states). Further, evidence of the phenomena is given in our 60 FPS hardware encoder. As can be seen, the encoding only increases the CPU usage by 5% but the energy consumption increases almost 16%.

Encoding	Before Streaming (FPS)	Streaming (FPS)
SW	85	65
HW	85	75

Table 2: Dota 2

From these experiments, it is clear that, in terms of both CPU and energy consumption, the hardware encoders are far superior. Whereas this looks intuitive, in the next section, we will show that due to *blocking calls*, hardware encoding can face significant issues in the whole pipeline.

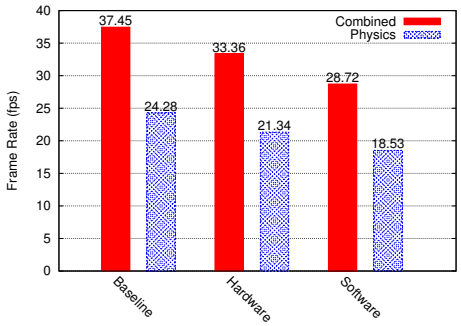

Figure 3: 3DMark Results Local Stream

3.3 Gaming Performance While Streaming

To determine the performance impacts that streaming has on gaming, we first utilize Futuremark's widely used gaming benchmark `3DMark`.[6] Specifically, we use the `Fire Strike` module, which is a 1080p benchmark utilizing the latest DirectX features (DX11) that stresses not only the 3D rendering capabilities of the system but also the physics calculations done on the CPU. Although 3D mark provides a proprietary score for comparing different systems, we are only interested in performance degradation due to recording/streaming. For this reason, we compare only the FPS lost between trials on two particular benchmarks. The first benchmark, the Fire Strike combined test, applies maximum stress to both the GPU and CPU. The second benchmark is the Physics benchmark, which has a performance limited due to the CPU. We again utilize OBS streaming 1080p video at 30 FPS, with a CBR of 3500 Kb/s using both hardware and software encoders.

The results for this experiment are given in Figure 3. Using the combined benchmarks, we experience a drop of just over 10% for hardware encoding. Further, if we are using the software x264 encoder, the combined benchmarks falls by over 23%. A similar story can be seen with the Physics test, where the performance of 3DMark falls by just over 12% when NVENC is utilized for encoding. The software encoding once again suffers the largest drop in performance falling by almost 24%. Based on our previous results on the CPU consumption of these two encoding methods, it is not surprising that x264 massive CPU requirements would interfere with the frame rate of a game benchmark.

Although synthetic benchmarks like 3DMark are great for running consistent experiments across different platforms,

[6] `http://www.3dmark.com/`

we expect to make sure our observations in 3DMark could be generalized to real games. To this end, we employed the use of the extremely popular game Dota 2 (Defense of the Ancients 2). Dota 2 makes a perfect candidate for our real world game test because not only does it utilize advanced graphics, it is also one of the most popular games to be streamed on Twitch, often attracting hundred's of simultaneous live streaming channels and tens of thousands of viewers. We set every detail level to maximum and also configured the game to run in ultra high definition (UHD) 4096 x 2160 resolution. Since OBS and Twitch do not currently support 4K streaming, we configure OBS to capture 4K and down sample to 1080p. We again utilize OBS optimal 1080p streaming settings of 30 FPS CBR of 3500 Kb/s. To benchmark the game, we find a high detail location on the map with stable FPS performance. We then enable streaming, first utilizing x264 software encoder and then using the hardware H.264 encoder. We stream to Twitch for one minute using both encoding methods and give the performance results in table 2. As can be seen, the results verify our earlier results from the 3DMark streaming experiment. Like the previous synthetic gaming benchmark, both software and hardware encoding methods cause measurable decreases in the game's performance. The software encoding reduced the performance of Dota 2 from 85 to 65 FPS, a drop of over 23%. The hardware encoding method did much better, dropping at only 12%.

4. PROFILING OBS

Beyond looking at performance and resource consumption issues, we also expect to improve the performance of the OBS streaming system. To that end, we profiled different stages of the encoding and streaming pipeline to determine finding significant issues with certain components. Referring back to Figure 1, we see that a streaming pipeline can be roughly broken down into 4 main stages: image capture, image conversion, encoding, and streaming. For each of these stages we calculate the average time in milliseconds taken to complete the stage. To determine which stages suffer latency increases with the resolution of the input frame we tested both 720p (1280x720) and 1080p (1920x1080), for both the hardware and software encoder. We again employ the Heaven benchmark to provide an image source, and we report the average time taken by each stage after a complete run of the benchmark, which is multiple thousand samples per run.

We give the results for the profiling in Table 3. As can be seen, some intuitive patterns emerge from the data, namely that as the input frame resolution increases, so does the amount of time needed to prepare the frame. Also, the average time to prepare and stream the frame stays small regardless of the encoding method or input frame size. However, hardware encoding seems to be significantly slower at both capturing the raw image source and converting the image from YUV444 to YUV420. For example, in the 1080p case, capturing the image takes nearly 75% longer when the hardware encoder is utilized. Also, at 1080p the colour conversion takes over 50% longer, when we are utilizing the hardware encoder. This is very surprising, given the fact that hardware and software encoding both perform this operation on the CPU using identical functions. In fact, by viewing the source code for OBS we find that the only difference between using the hardware and software encoding should come from the encoding phase, since that is the only module in the pipeline directly changed by choosing either hardware or software encoders. The latency of this pipeline is of critical importance, especially when streaming at higher frame rates. For example, at 60 FPS streaming a new frame should exit the pipeline approximately every 16 ms in order to provide a stable video stream. We will discuss what causes this increased latency, and how to optimize it in the next section.

4.1 Optimizing Hardware Encoding

Through our profiling of OBS and paying particular attention to the CPU sleep states, we find that when the software encoder enters the convert image phase seen in Figure 1 all cores are active and running at their max clock speed. This is because all 4 cores on our test platform were just engaged in encoding the raw image into a compressed H.264 stream, a very CPU intensive operation. However, when the hardware encoder enters convert phase only 2 of the cores are running at their maximum performance. We have discovered this is because the call to the hardware encoder is a *blocking call*; that is, once OBS encodes a frame utilizing the hardware, it can sleep for over 10 ms. Due to this period of low utilization, the operating system and processor respond by down clocking the CPU cores and putting some of them to sleep to conserve energy. As such, when OBS reaches the image conversion phase the cores are in their energy conservation state, which creates a lag time before they are running again at high performance.

We present two possible solutions to this issue. The naive solution is to completely disable power management. Although, this will alleviate the issue by forcing the cores to constantly be at maximum performance, we will lose all the power saving advantages of using a hardware encoder. We propose a more elegant solution of advertising to the operating system that we will be needing the maximum performance of the CPU before we enter the image conversion section and then restoring the power state after converting the colours. We have implemented this idea in OBS and the results are given in Table 4[7]. When compared to the previous encoding latency results in Table 3, we see that our modification has greatly improved the encoding efficiency of the hardware encoder by decreasing the latency of both the capture-image and convert-image phase of the OBS pipeline. Further, our initial power measurement indicates that our solution only increase the power consumption by approximately 32 watts to 190 watts while recording using the hardware encoder, which is much more efficient then the software encoders performance at 250 watts.

5. BRIDGING CLOUD GAMING AND LIVE BROADCASTING

The last topic we explored is how to bridge a real-world cloud gaming system. Bridging cloud gaming and game streaming has a number of appealing features for both developers and gamers. First, residential broad-band connection often offer asymmetric download and upload speeds, with the upload speeds being severely limited. For example, according to OOKLA[8] the average uplink speed in

[7]Code Available:`http://www.sfu.ca/~rws1/game-stream/`
[8]Net Index:`http://www.netindex.com/upload/`

Encoding	Capture-Image	Convert-Image	Encode Frame	Stream Frame	Total OBS
SW 720p	6.4 ms	4.4 ms	2.8 ms	0.2 ms	13.8 ms
SW 1080p	11.0 ms	4.9 ms	3.0 ms	0.3 ms	19.2 ms
HW 720p	7.2 ms	6.5 ms	12.4 ms	0.2 ms	26.3 ms
HW 1080p	19.3 ms	7.5 ms	16.4 ms	0.3 ms	43.5 ms

Table 3: Timing across OBS pipeline

Encoding	Capture-Image	Convert-Image	Encode Frame	Stream Frame	Total OBS
HW 720p (opt)	5.2 ms	2.8 ms	11.8 ms	0.2 ms	20.0 ms
HW 1080p (opt)	9.6 ms	2.5 ms	14.1 ms	0.1 ms	26.3 ms

Table 4: Improved Latency Hardware Encoding

Germany is 4.1 Mb/s, Canada is 6.6 Mb/s and the USA is 9.6 Mb/s. The low average uplink speeds of many countries are a serious impediment to streaming higher quality FHD or for future streaming at UHD. Also, with such low up uplink speeds, game streaming could interfere with the players QoE while playing online games due to network congestion, even at twitches recommended 3.5 Mb/s. Further, our profiling in section 4 indicates that OBS can have issues streaming at seriously effecting the games performance.

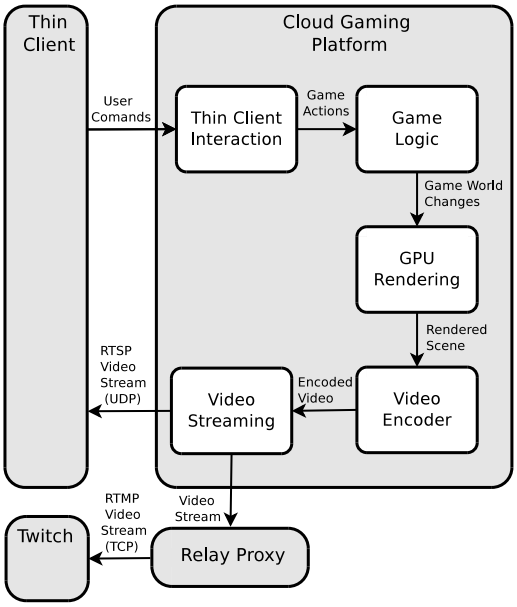

Figure 4: Cloud Gaming With Live Streaming Support

5.1 System Overview

Our research group has implemented a fully virtualized cloud gaming system called Rhizome. Next, we briefly introduce Rhizome and its architecture.[9] A sketch of the Rhizome's architecture extended to stream to twitch is given in Figure 4.

The first module is the *Game Logic*, which in essence is the gaming application that the user is playing. Together with the *Thin Client Interaction* module, it intercepts the user's keystrokes sent from the remote client to the game and computes the game world changes. The rendering is then performed by the GPU that is assigned to the VM, e.g., an NVIDIA GRID GK104. The rendered scenes are passed to the *Video Encoder* module that contains a video encoder and a discrete framer. The video encoder, which we will highlight later, is selectable, consisting of either a software or hardware H.264 encoder. In either case, the encoder needs additional support to be adapted for real-time streaming, namely, a discreet framer, which allows the Live555 streaming library to request live frames from the encoders at a desired video stream frame rate. The encoded video stream is then encapsulated and transported in a UDP RTSP stream to our think client.

Our design and implementation are platform-independent, although a NVIDIA GRID GPU is required for our hardware encoding implementation. We have deployed and experimented the system on Amazon EC2 GPU Instances (G2), a new type of cloud instances backed by the Intel Xeon E5-2670 (Sandy Bridge) processors and the NVIDIA GRID-K520 board that contains a GK104 GPU with 1536 CUDA cores and 4GB of video memory. The GRID's on-board hardware video encoder supports up to 8 live HD video streams (720p at 30 fps) or up to 4 live FHD video streams (1080p at 30 fps), as well as low-latency frame capture for either the entire screen or selected rendering objects, enabling a G2 instance to offer high-quality interactive streaming such as 3D game streaming.

We modify our gaming platform to provide a secondary video stream to a new module, the relay proxy. The relay proxy converts the video stream to the encapsulation format specified by a streaming service, in this case RTMP over TCP for Twitch. We implement the relay proxy using FFMPEG, which can either run directly on our cloud gaming virtual machine or, for maximum performance, on its own VM. The proxy has a number of additional usages such as combining multiple video streams together, such as the game stream and the thin clients web-cam, as well as transcoding to different formats. Further, thanks to the large bandwidth available in EC2 (up to 1000 Mb/s on a EC2 large instance) the proxy itself could distribute the content to many viewers.

5.2 Performance

To determine the performance impacts that live game streaming has on gaming we again utilize Futuremark's widely used gaming benchmark `3DMark`. For a comparison between our modified cloud gaming platform and traditional game streaming we utilize OBS streaming 1080p video at 30 FPS, with a CBR of 3500 Kb/s. We also compute a base-line performance of 3DMark running on our platform with all streaming and encoding tasks disabled. Although disabling all video outputs from our platform implies that you would have to physically be in the data-centre with a

[9]More Info:`http://www.sfu.ca/~rws1/cloud-gaming/`

monitor to see the game, it does provide a useful best case scenario for what the maximum gaming performance of the EC2 instance is.

The results are given in Figure 5. When we supply a live gaming stream from our cloud platform using OBS gaming performance drops from 26.28 to 25.16 FPS, a measurable drop of just over 8%. The impact on game physics caused by OBS is much larger dropping nearly 25%. Our cloud gaming implementation on the other hand suffers a performance drop of less than 2% to either its combined or physics performance. The reason our cloud gaming platform has such a large advantage over OBS is mainly to do to our implementation of the capture, convert, and encode stage of the rhizome pipeline. In OBS each of these stages involve a copy from the GPU, converting using a set of threads, and finally encoding using x264 or NVENC. However, we combine these into a single stage utilizing the GPU for capture, conversion, and encoding. Thus, unlike OBS we only need a single copy from the GPU in order to receive a fully encoded H.264 frame. We have profiled Rhizome and find that the average latency of this phase takes on average less than 12 ms to complete, which is 39.2% faster than our best case of OBS we tested. Further, our analysis shows that Rhizome only consumes 10% of the CPU leaving more resources available for the gaming process.

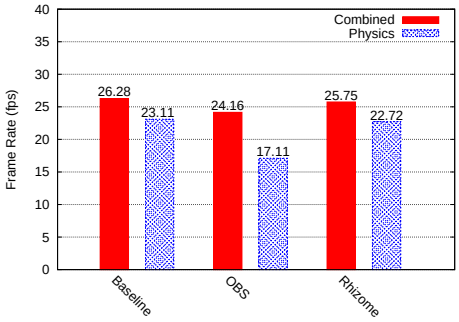

Figure 5: 3DMark Results Cloud Game Stream

6. RELATED WORK

Recently there have been some significant studies surrounding issues related to live game streaming. Pires *et al.* [8] discuss enhancing Twitch with DASH to improve QoE and reduce infrastructure costs. Others have explored the design issues and implications of participatory mixed-media environments such as Twitch [3][6].

Further studies have been performed to understand the user-perceived QoE in cloud gaming systems, Jarschel *et al.* [5] conducted a subjective study, in which the selected participants were asked to play videogames of slow, medium, and fast gameplay through their setup under different latency and packet-loss conditions. A recent study from Claypool *et al.* [2] provided a detailed study of OnLive, a commercially available cloud gaming system, and closely analyzed its bitrates, packet sizes and inter-packet times for both upstream and downstream game traffic. To further clarify its streaming quality, Shea *et al.* [9] measured the real-world performance of OnLive with different types of network and bandwidth conditions. The authors carefully studied the streaming quality and revealed critical challenges toward the widespread deployment of cloud gaming. Wu *et al.* [10] further conducted a series of passive measurements on a large-scale cloud gaming platform and identified the performance issues of queueing delay as well as response delay among users. Huang *et al.* [4] provided an open-source cloud gaming system *GamingAnywhere*, which has been deployed on the Android OS with extensive experiments performed.

7. CONCLUSION

In this paper we have systematically investigated game streaming, both with traditional CPU based encoders and more advanced hardware encoding devices. We found that in general hardware encoders can greatly reduce both the energy usage and performance implication of live game streaming. Our profiling of these systems shows that due to some interesting interplay between the CPU and GPU based hardware encoders some practical optimizations can be made to greatly decrease the encoding latency. Finally, we took the first steps towards implementing live streaming to many viewers in a cloud gaming context. Our prototype implementation indicates that combining these platforms is not only feasible but also results in nearly negligible performance degradation while live streaming.

8. REFERENCES

[1] Sizing and profiling esports popularity. `http://www.newzoo.com/insights/free-report-sizing-profiling-esports-popularity/`. Accessed: 2014-12-8.

[2] M. Claypool and D. Finkel. On the performance of onlive thin client games. *Springer Multimedia Systems Journal, Special Issue on Network Systems Support for Games*, PP(9):1–14, 2014.

[3] W. A. Hamilton, O. Garretson, and A. Kerne. Streaming on twitch: fostering participatory communities of play within live mixed media. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 1315–1324. ACM, 2014.

[4] C. Huang, C. Hsu, D. Chen, and K. Chen. Quantifying user satisfaction in mobile cloud games. *in Proceedings of ACM MoViD*, 2014.

[5] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hossfeld. Gaming in the clouds: Qoe and the users' perspective. mathematical and computer modelling. *ACM Transactions on Computer Systems*, 57(11):2883–2894, 2013.

[6] M. Kaytoue, A. Silva, L. Cerf, W. Meira Jr, and C. Raïssi. Watch me playing, i am a professional: a first study on video game live streaming. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 1181–1188. ACM, 2012.

[7] Matthew DiPietro. Twitch Hits One Million Monthly Active Broadcasters. http://blog.twitch.tv/2014/02/twitch-hits-one-million-monthly-active-broadcasters/.

[8] K. Pires and G. Simon. Dash in twitch: Adaptive bitrate streaming in live game streaming platforms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 13–18. ACM, 2014.

[9] R. Shea and J. Liu. Cloud gaming: Architecture and performance. *IEEE Network*, 27(4):16–21, 2013.

[10] D. Wu, Z. Xue, and J. He. icloudaccess: Costeffective streaming of video games from the cloud with low latency. *IEEE Transactions on Circuits and Systems for Video Technology*, PP(99):1–12, 2014.